Kapitel 6: Daten laden, speichern und Dateiformate

McKinney, W. (2017). *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. 2. Auflage. Sebastopol, CA [u. a.]: O'Reilly.

Überarbeitet: armin.baenziger@zhaw.ch, 16. Januar 2020

- Bevor Daten analysiert werden können, müssen sie geladen werden.
- (Aufbereitete) Daten müssen gespeichert werden können, damit sie später (von anderen) wieder verwendet werden können.
- In diesem Kapitel werden einige Möglichkeiten in Pandas vorgestellt, Daten in unterschiedlichen Formaten zu laden und speichern.
- Das Lehrmittel geht in diesem Kapitel deutlich weiter. Insbesondere werden dort auch noch Interaktionen mit *Web APIs* und *Datenbanken* vorgestellt.

In [1]:
```
%autosave 0
```

```
Autosave disabled
```

In [2]:
```
# Nötige Bibliotheken mit üblichen Abkürzungen laden:
import numpy as np
import pandas as pd
```

Daten im Textformat lesen und schreiben

Wir beginnen mit dem Einlesen von einer kleinen Komma-separierten (CSV) Text-Datei. Diese sieht wie folgt aus:

In [ ]:
```
# %load ../weitere_Daten/Beispiele_Kap06/CSV_Komma.csv
Name, Vorname, Geschlecht, Gehalt
Brunner, Sandra, w, 5600
Lonza, Daniele, m,
Zender, Thomas, m, 5500
```

- CSV-Daten kann man sehr einfach mit der Pandas-Funktion `read_csv` einlesen.
- Wenn es Probleme mit dem Einlesen gibt, stimmt der Pfad evt. nicht. Der Pfad muss zur Datei `CSV_Komma.csv` im Ordner `weitere_Daten/Beispiele_Kap06` führen.
- Die zwei Punkte am Anfang des Pfads bedeuten, dass `read_csv` zuerst eine Hierarchiestufe im Ordnerpfad hoch geht und dann im Ordner `weitere_Daten/Beispiele_Kap06` die Datei `CSV_Komma.csv` versucht zu öffnen.

In [4]:
```
Lohndaten = pd.read_csv('../weitere_Daten/Beispiele_Kap06/CSV_Komma.csv')
Lohndaten
```

Out[4]:

	Name	Vorname	Geschlecht	Gehalt
0	Brunner	Sandra	w	5600.0
1	Lonza	Daniele	m	NaN
2	Zender	Thomas	m	5500.0

Die allgemeinere Funktion `pd.read_table` mit Angabe des *Trennzeichens (engl. delimiter)* hätte auch zum gewünschten Resultat geführt.

In [5]:
```
pd.read_table('../weitere_Daten/Beispiele_Kap06/CSV_Komma.csv', sep=',')
```

Out[5]:

	Name	Vorname	Geschlecht	Gehalt
0	Brunner	Sandra	w	5600.0
1	Lonza	Daniele	m	NaN
2	Zender	Thomas	m	5500.0

Zeilen mit Fehlwerten ganz weglassen:

In [6]:
```
Lohndaten.dropna(inplace=True)
Lohndaten
```

Out[6]:

	Name	Vorname	Geschlecht	Gehalt
0	Brunner	Sandra	w	5600.0
2	Zender	Thomas	m	5500.0

CSV-Datei mit anderen Trennzeichen:

In [ ]:
```
# %load ../weitere_Daten/Beispiele_Kap06/CSV_Strichpunkt.csv
Name; Vorname; Geschlecht; Gehalt
Brunner; Sandra; w; 5600
Lonza; Daniele; m;
Zender; Thomas; m; 5500
```

In [8]:
```
Lohndaten = pd.read_csv('../weitere_Daten/Beispiele_Kap06/CSV_Strichpunkt.csv',
                        sep=';')    # mit dem Argument sep andere Trennzeichen an
geben
Lohndaten
```

Out[8]:

	Name	Vorname	Geschlecht	Gehalt
0	Brunner	Sandra	w	5600.0
1	Lonza	Daniele	m	NaN
2	Zender	Thomas	m	5500.0

CSV-Datei ohne Header laden:

In [ ]:
```
# %load ../weitere_Daten/Beispiele_Kap06/CSV_KeinHeader.csv
Brunner, Sandra, w, 5600
Lonza, Daniele, m,
Zender, Thomas, m, 5500
```

In [10]:
```
Lohndaten2 = pd.read_csv('../weitere_Daten/Beispiele_Kap06/CSV_KeinHeader.csv',
                         header=None)    # Kopfzeile fehlt
Lohndaten2
```

Out[10]:

	0	1	2	3
0	Brunner	Sandra	w	5600.0
1	Lonza	Daniele	m	NaN
2	Zender	Thomas	m	5500.0

Entweder man fügt nun die Variablennamen ein oder man lädt die Datei wie folgt:

In [11]:
```
Lohndaten2 = pd.read_csv('../weitere_Daten/Beispiele_Kap06/CSV_KeinHeader.csv',
                         header=None,    # Kopfzeile fehlt
                         names=['Name', 'Vorname', 'Geschlecht', 'Gehalt'])
Lohndaten2
```

Out[11]:

	Name	Vorname	Geschlecht	Gehalt
0	Brunner	Sandra	w	5600.0
1	Lonza	Daniele	m	NaN
2	Zender	Thomas	m	5500.0

Exkurs: Textdateien ohne fixes Trennzeichen:

In [ ]:
```
# %load ../examples/ex3.txt
            A         B         C
aaa -0.264438 -1.026059 -0.619500
bbb  0.927272  0.302904 -0.032399
ccc -0.264273 -0.386314 -0.217601
ddd -0.871858 -0.348382  1.100491
```

Die Felder werden hier durch (unterschiedlich viele) Leerzeichen getrennt.

In [13]:
```
df = pd.read_table('../examples/ex3.txt', delim_whitespace=True)
df
```

Out[13]:

	A	B	C
aaa	-0.264438	-1.026059	-0.619500
bbb	0.927272	0.302904	-0.032399
ccc	-0.264273	-0.386314	-0.217601
ddd	-0.871858	-0.348382	1.100491

Ende des Exkurses.

Mit **Fehlwerten** umgehen:

In [ ]:
```
# %load ../examples/ex5.csv
something,a,b,c,d,message
one,1,2,3,4,NA
two,5,6,,8,world
three,9,10,11,12,foo
```

In [15]:
```
df = pd.read_csv('../examples/ex5.csv')
df
```

Out[15]:

	something	a	b	c	d	message
0	one	1	2	3.0	4	NaN
1	two	5	6	NaN	8	world
2	three	9	10	11.0	12	foo

In [16]:
```
df.isnull()     # Welche Zellen werden als Fehlwerte erkannt?
```

Out[16]:

	something	a	b	c	d	message
0	False	False	False	False	False	True
1	False	False	False	True	False	False
2	False	False	False	False	False	False

In [17]:
```
df.isnull().sum()    # Anzahl Fehlwerte pro Spalte
```

Out[17]:
```
something    0
a            0
b            0
c            1
d            0
message      1
dtype: int64
```

Mit dem Argument `na_values` kann man (weitere) Fehlwerte angeben. Angenommen *foo* ist eine Bezeichnung für einen (weiteren) Fehlwert:

In [18]:
```
df = pd.read_csv('../examples/ex5.csv', na_values=['foo'])
df
```

Out[18]:

	something	a	b	c	d	message
0	one	1	2	3.0	4	NaN
1	two	5	6	NaN	8	world
2	three	9	10	11.0	12	NaN

Daten im Textformat schreiben

Wir werden nun den eben verwendeten Datensatz im Textformat abspeichern. Zuerst setzen wir noch den Index neu, um zu zeigen, dass der Index auch mit abgespeichert wird.

In [19]:
```
df.set_index('something')
```

Out[19]:

	a	b	c	d	message
something					
one	1	2	3.0	4	NaN
two	5	6	NaN	8	world
three	9	10	11.0	12	NaN

Jetzt schreiben wir diese Daten im CSV-Format ins Unterverzeichnis "examples". Wir bezeichnen die Datei mit "out.csv".

In [20]:
```
df.to_csv('../examples/out.csv')
```

In [ ]:
```
# %load ../examples/out.csv
,something,a,b,c,d,message
0,one,1,2,3.0,4,
1,two,5,6,,8,world
2,three,9,10,11.0,12,
```

Man kann die Daten auch mit anderem Trennzeichen schreiben.

In [22]:
```
df.to_csv('../examples/out.csv', sep='|')  # mit anderem Trennzeichen
```

In [ ]:
```
# %load ../examples/out.csv
|something|a|b|c|d|message
0|one|1|2|3.0|4|
1|two|5|6||8|world
2|three|9|10|11.0|12|
```

Mit dem Argument `na_rep` können Fehlwerte (anders) gekennzeichnet werden.

In [24]:
```
df.to_csv('../examples/out.csv', na_rep='999')
```

In [ ]:
```
# %load ../examples/out.csv
,something,a,b,c,d,message
0,one,1,2,3.0,4,999
1,two,5,6,999,8,world
2,three,9,10,11.0,12,999
```

Kontrollfrage:

In [26]:
```
# Laden Sie die Datei "CSV_Kontrollfrage.csv" vom Ordner
# "weitere_Daten/Beispiele_Kap06" in das DataFrame "df".
# Prüfen Sie (z. B. mit einem Text-Editor) zuerst, welches
# Trennzeichen in der Textdatei verwendet wird.
df = pd.read_csv('../weitere_Daten/Beispiele_Kap06/CSV_Kontrollfrage.csv',
                 sep=';')
df.head()
```

Out[26]:

	VariableA	VariableB
0	4	6
1	0	2
2	-1	5

Binäre Datenformate

Einfach und effizient lassen sich Daten in Python im **'pickle'-Format** speichern und lesen.

In [27]:
```
df = pd.read_csv('../examples/ex1.csv')
df
```

Out[27]:

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

In [28]:
```
# DataFrame im pickle-Format speichern:
df.to_pickle('../examples/frame_pickle.pkl')
```

Achtung: Das pickle-Format eigenet sich nicht für die *Langzeitspeicherung* der Daten, da sich das Dateiformat ändern kann ich künftigen Library-Versionen.

In [29]:
```
# DataFrame im pickle-Format lesen:
df = pd.read_pickle('../examples/frame_pickle.pkl')
df
```

Out[29]:

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

Microsoft Excel Dateien einlesen

Falls die Daten in Microsoft Excel im CSV-Format gespeichert wurden (was empfehlenswert ist), verwendet man die Funktionen, welche wir eben kennengelernt haben.

Daten im proprietären Excel-Format können aber auch eingelesen werden:

In [30]:
```
# Default: Erstes Excel-Blatt (Sheet) öffnen:
df = pd.read_excel('../examples/ex1.xlsx')
df
```

Out[30]:

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

In [31]:
```
# Ein anderes Excel-Blatt einlesen (hier Sheet2):
df = pd.read_excel('../examples/ex1.xlsx', 'Sheet2')
df
```

Out[31]:

	A	B	C	D	Meldung
0	11	12	13	14	Hallo
1	15	16	17	18	Welt
2	19	20	21	22	Platzhalter

Daten im Excel-Format schreiben:

In [32]:
```
df.to_excel('temp.xlsx')
```

In [33]:
```
# Exkurs:
import os                # Bibliothek für Betriebssysteminteraktionen
os.remove('temp.xlsx')   # Datei wieder löschen auf der Festplatte.
```

Fazit

- Der Zugriff auf Daten ist häufig der erste Schritt im Datenanalyseprozess.
- Wir haben uns in diesem Kapitel einige nützliche Tools angesehen, die Ihnen den Einstieg erleichtern sollen.
- In den folgenden zwei Kapiteln werden wir uns eingehender mit Daten-Wrangling befassen.